

# **MULTI: Configuring Connections for PowerPC Targets**



**Green Hills Software, Inc.**

**30 West Sola Street  
Santa Barbara, California 93101  
USA**

**Tel: 805-965-6044  
Fax: 805-965-6343  
[www.ghs.com](http://www.ghs.com)**

## **DISCLAIMER**

GREEN HILLS SOFTWARE, INC. MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. Further, Green Hills Software, Inc. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Green Hills Software, Inc. to notify any person of such revision or changes.

Copyright © 1983-2009 by Green Hills Software, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission from Green Hills Software, Inc.

Green Hills, the Green Hills logo, CodeBalance, GMART, GSTART, INTEGRITY, MULTI, and Slingshot are registered trademarks of Green Hills Software, Inc. AdaMULTI, Built with INTEGRITY, EventAnalyzer, G-Cover, GHnet, GHnetLite, Green Hills Probe, Integrate, ISIM, u-velOSity, PathAnalyzer, Quick Start, ResourceAnalyzer, Safety Critical Products, SuperTrace Probe, TimeMachine, TotalDeveloper, DoubleCheck, and velOSity are trademarks of Green Hills Software, Inc.

All other company, product, or service names mentioned in this book may be trademarks or service marks of their respective owners.

PubID: 10685  
April 25, 2014

# Contents

---

|          |                                                                                  |      |
|----------|----------------------------------------------------------------------------------|------|
|          | <b>Preface</b>                                                                   | iii  |
|          | About This Book                                                                  | iv   |
|          | Supported Target Connections for PowerPC                                         | v    |
|          | The MULTI 5 Document Set                                                         | vii  |
|          | Conventions Used in the MULTI Document Set                                       | viii |
| <b>1</b> | <b>Introduction</b>                                                              | 1    |
|          | The MULTI Integrated Development Environment                                     | 2    |
|          | MULTI Launcher                                                                   | 2    |
|          | Editing Tools                                                                    | 2    |
|          | Building Tools                                                                   | 2    |
|          | Debugging Tools                                                                  | 3    |
|          | Administrative Tools                                                             | 3    |
|          | The MULTI Launcher                                                               | 4    |
|          | MULTI Workspaces                                                                 | 5    |
|          | Accessing Online Help                                                            | 6    |
|          | Full Online Manuals                                                              | 6    |
|          | Context-Sensitive Help                                                           | 6    |
|          | Command and Configuration Option Help                                            | 6    |
|          | The Help Viewer Window                                                           | 7    |
|          | Navigating Manuals in the Help Viewer                                            | 8    |
| <b>2</b> | <b>Setting Up Your Target Hardware</b>                                           | 13   |
|          | Installing Your Target Hardware                                                  | 14   |
|          | Configuring Your Target Hardware                                                 | 14   |
|          | Using the Project Wizard to Configure your<br>Debugging Environment              | 14   |
|          | Testing and Customizing your Debugging<br>Environment                            | 17   |
|          | Specifying Setup Scripts                                                         | 31   |
|          | Using MULTI (.mbs) Setup Scripts When<br>Connecting to Your Target               | 32   |
|          | Using Legacy Debug Server (.dbs) Setup Scripts<br>When Connecting to Your Target | 33   |
|          | Running Setup Scripts Manually                                                   | 33   |

# Contents

---

|          |                                                                 |           |
|----------|-----------------------------------------------------------------|-----------|
| <b>3</b> | <b>Cafe OS (cafeserv) Connections</b>                           | <b>37</b> |
|          | Connecting to Your Target Running Cafe OS                       | 38        |
|          | Creating a Standard Cafe OS (cafeserv) Connection               |           |
|          | Method                                                          | 38        |
|          | Using the Cafe OS (cafeserv) Connection Editor                  | 39        |
| <b>A</b> | <b>Green Hills Debug Server Command and Scripting Reference</b> | <b>45</b> |
|          | Green Hills Debug Server Commands                               | 46        |
|          | Generic Debug Server Commands                                   | 46        |
|          | Additional Debug Server Commands                                | 52        |
|          | The Green Hills Debug Server Scripting Language                 | 53        |
|          | General Notes                                                   | 53        |
|          | Scripting Syntax                                                | 54        |
|          | Example Scripts                                                 | 57        |
|          | <b>Index</b>                                                    | <b>61</b> |

# Preface

---

*This Preface Contains:*

- About This Book
- The MULTI 5 Document Set
- Conventions Used in the MULTI Document Set

This section discusses the purpose of the manual, typographical conventions used, and the MULTI documentation set.

## About This Book

---

The MULTI Debugger can be used to debug applications that reside on native, embedded, or simulated *targets*. To debug these applications, MULTI must first connect to the target through a *debug server* that runs on the host. Each distribution of the MULTI Integrated Development Environment (IDE) includes multiple debug servers, each of which supports connections to a specific set of probes, in-circuit emulators, monitors, and/or target simulators. This book describes how to set up your specific debugging interface for use with MULTI and how to configure connections to your target using the appropriate debug server. Specifically:

- Chapter 2, “Setting Up Your Target Hardware”, explains procedures you may need to perform if you are using an embedded hardware target. In such cases, you may need to configure your embedded target using a target setup script. MULTI provides default setup scripts for most supported targets. Chapter 2 explains how to access these setup scripts and, if necessary, how to customize them for your particular target.
- Chapters 3-9 each provide connecting instructions for the targets supported by a specific debug server. Because each of the target systems supported by MULTI has different features and limitations, the debug servers that support connections with these various targets also have different features, commands, and options, which are documented in chapters 3-9.
- Appendix A, “Green Hills Debug Server Command and Scripting Reference”, documents the commands that can be issued to Green Hills debug servers and describes the scripting language and conventions used for writing target setup scripts in previous versions of MULTI. The scripting language described in this appendix has been deprecated beginning with MULTI 5.0. The new scripting conventions are described in Chapter 2, “Using MULTI Scripts” in the *MULTI: Scripting* book.



**Note** The general procedure for connecting to targets using MULTI’s debug servers is described in Chapter 4, “Connecting to Your Target” in the *MULTI: Debugging* book. The present book supplements that chapter with specific instructions about configuring the debug servers and targets that are supported

for PowerPC processors. (See the next section for a list of these supported targets.)



**Note** In this book, the word *target* indicates either an embedded or a simulated target. If you are developing in a native environment, MULTI generally “connects” transparently through a simple debug server. Because this happens automatically and because there are usually no configuration options that can be set for such connections, native connections are not discussed in this book. If there are useful connection options available for your particular native connection type, they will be documented in the *MULTI: Building Applications* book for your specific native environment.



**Note** New or updated information may have become available while this book was in production. For additional material that was not available at press time, or for revisions that may have become necessary since this book was printed, please check your MULTI installation directory for release notes, **README** files, and other supplementary documentation.

## Supported Target Connections for PowerPC

The following table lists the target connections supported for PowerPC. For detailed information about each type of target connection, see the appropriate chapter, as identified in the table.

| Debugging interface or simulator | Green Hills debug server | Debug server chapter                        |
|----------------------------------|--------------------------|---------------------------------------------|
| cafeserv                         | cafeserv                 | Chapter 3, “Cafe OS (cafeserv) Connections” |



**Note** This book does not document how to make connections to OS targets or use MULTI to perform OS debugging. If you are using MULTI with the Green Hills INTEGRITY RTOS, see the INTEGRITY document set you received with your INTEGRITY distribution. If you are using another supported OS target, see the appropriate book from the list below.

- *MULTI: Developing for ThreadX*
- *MULTI: Developing for VxWorks*

This list may not be comprehensive. Contact Green Hills Technical Support if your particular debugging interface, monitor, simulator, or OS does not appear in the list.



---

## The MULTI 5 Document Set

---

The primary documentation for using MULTI is provided in the following books:

- *MULTI: Getting Started* — Describes how to install MULTI, and takes you through creating, building, and debugging an example project.
- *MULTI: Licensing* — Describes how to obtain, install, and administer Green Hills licenses. This book is intended for system administrators.
- *MULTI: Editing Files and Configuring the IDE* — Describes how to use the MULTI Editor and MULTI Launcher, how to use a version control system with MULTI, and how to configure the MULTI IDE.
- *MULTI: Building Applications* — Describes how to use the MULTI Project Manager and compiler drivers and the tools that compile, assemble, and link your code. Also describes the Green Hills implementation of supported high-level languages.
- *MULTI: Configuring Connections* — Describes how to set up your target debugging interface for use with MULTI and how to configure connections to your target.
- *MULTI: Debugging* — Describes how to use the MULTI Debugger and its associated tools.
- *MULTI: Debugging Command Reference* — Describes how to use Debugger commands and provides a comprehensive reference of Debugger commands.

For a comprehensive list of the books provided with your MULTI installation, see the **toc\_target.pdf** file located in the **manuals** subdirectory of your installation.

Most books are available in the following formats:

- A printed book (select books are not available in print).
- Online help, accessible from most MULTI windows via the **Help** → **Manuals** menu.
- An electronic PDF, available in the **manuals** subdirectory of your installation.

## Conventions Used in the MULTI Document Set

---

All Green Hills documentation assumes that you have a working knowledge of your host operating system and its conventions, including its command line and graphical user interface (GUI) modes. For example, you should know how to use basic commands, how to open, save, and close files, and how to use a mouse and standard menus.

Green Hills documentation uses a variety of notational conventions to present information and describe procedures. These conventions are described below.

| Convention         | Meaning                                                                                                                                                                                                                     | Example                                                                                                                                                                                                                                                                        |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>bold type</b>   | Indicates a: <ul style="list-style-type: none"><li>• Filename or pathname</li><li>• Command</li><li>• Option</li><li>• Window title</li><li>• Menu name or menu choice</li><li>• Field name</li><li>• Button name</li></ul> | <ul style="list-style-type: none"><li>• <b>C:\MyProjects</b></li><li>• <b>setup</b> command</li><li>• <b>-G</b> option</li><li>• The <b>Breakpoints</b> window</li><li>• The <b>File</b> menu</li><li>• <b>Working Directory:</b></li><li>• The <b>Browse</b> button</li></ul> |
| <i>italic type</i> | Indicates: <ul style="list-style-type: none"><li>• Replaceable text</li><li>• A new term</li><li>• A book title</li></ul>                                                                                                   | <ul style="list-style-type: none"><li>• <b>-o</b> <i>filename</i></li><li>• A task may be called a <i>process</i> or a <i>thread</i></li><li>• <i>MULTI: Debugging</i></li></ul>                                                                                               |

| Convention                                             | Meaning                                                                                                                                                                                                                        | Example                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| monospace type                                         | Indicates: <ul style="list-style-type: none"> <li>• Text you should enter as presented</li> <li>• A word or words used in a command or example</li> <li>• Source code</li> <li>• Input/output</li> <li>• A function</li> </ul> | <ul style="list-style-type: none"> <li>• Type <code>help</code><br/><code>command_name</code></li> <li>• The <b>wait</b> [-global] command blocks command processing, where -global blocks command processing for all MULTI processes.</li> <li>• <code>int a = 3;</code></li> <li>• <code>&gt; print Test</code><br/><code>Test</code></li> <li>• <code>GHS_System()</code></li> </ul> |
| ellipsis (...) (in command line instructions)          | Indicates that the preceding argument or option can be repeated zero or more times.                                                                                                                                            | <b>debugbutton</b> [name]...                                                                                                                                                                                                                                                                                                                                                            |
| greater than sign ( > )                                | Represents a prompt. Your actual prompt may be a different symbol or string. The > prompt helps to distinguish input from output in examples of screen displays.                                                               | <code>&gt; print Test</code><br><code>Test</code>                                                                                                                                                                                                                                                                                                                                       |
| pipe (   ) (in command line instructions)              | Indicates that one (and only one) of the parameters or options separated by the pipe or pipes should be specified.                                                                                                             | <b>call</b> <i>proc</i>   <i>expr</i>                                                                                                                                                                                                                                                                                                                                                   |
| square brackets ( [ ] ) (in command line instructions) | Indicate optional arguments, commands, options, and so on. You can either include or omit the enclosed elements. The square brackets should not appear in your actual command.                                                 | <code>.macro</code> <i>name</i> [ <i>list</i> ]                                                                                                                                                                                                                                                                                                                                         |

The following command description demonstrates the use of some of these typographical conventions.

**gxyz** [-option]... *filename*

The formatting of this command indicates that:

- The command **gxyz** should be entered as shown.
- The option *-option* should either be replaced with one or more appropriate options or be omitted.
- The word *filename* should be replaced with the actual filename of an appropriate file.

The square brackets and the ellipsis should not appear in the actual command you enter.

# Introduction

---

## Chapter 1

*This Chapter Contains:*

- The MULTI Integrated Development Environment
- The MULTI Launcher
- Accessing Online Help

This chapter provides an overview of the MULTI Integrated Development Environment, and then gives a brief introduction to this book, which describes how to set up your specific debugging interface for use with MULTI and how to configure connections to your PowerPC target.

## The MULTI Integrated Development Environment

---

MULTI is a complete Integrated Development Environment (IDE) designed especially for embedded systems engineers to assist them in compiling, optimizing, debugging, and editing embedded applications.

The MULTI IDE includes the following tools for each part of the software development process. Many of the MULTI tools can be launched from within the IDE or as separate stand-alone programs. Parenthetical text indicates corresponding executable files, which are located in your MULTI installation.

### MULTI Launcher

**MULTI Launcher (mstart)** — The gateway to the MULTI IDE, which allows you to quickly launch any of the primary MULTI tools, access open windows, and manage MULTI workspaces.

### Editing Tools

- **MULTI Editor (me)** — A graphical editor for modifying text files.
- **Checkout Browser (mcobrowse)** — A graphical viewer for files managed under a version control system.
- **Diff Viewer (diffview)** — A graphical viewer that displays differences between two text files.

### Building Tools

- **MULTI Project Manager (mprojmgr)** — A graphical interface for managing and building projects.
- **Integrate (integrate)** — A graphical utility for configuring tasks, connections, and kernel objects across multiple address spaces when using the INTEGRITY RTOS.

## Debugging Tools

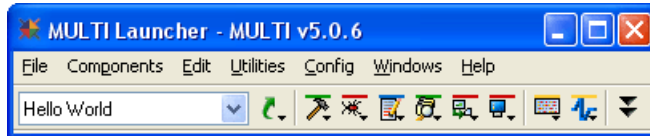
- **MULTI Debugger (multi)** — A graphical source-level debugger.
- **Instruction Set Simulator** — A tool that simulates your embedded processor on your development host.
- **EventAnalyzer (mevgui)** — A graphical viewer for monitoring the complex real-time interactions of an embedded RTOS such as INTEGRITY or u-velOSity.
- **ResourceAnalyzer (wperf)** — A graphical viewer for monitoring the CPU and memory usage of an embedded system running the INTEGRITY RTOS.
- **Serial Terminal (mterminal)** — A serial terminal emulator for connecting to serial ports on embedded devices.
- **DoubleCheck** — A static source code analysis tool for locating programming problems that lead to security and reliability failures in software.
- **MULTI TimeMachine Tool Suite** — A wide variety of trace analysis tools that dramatically extend MULTI's trace and debugging capabilities.

## Administrative Tools

- **Bug Report (gbugrpt)** — A utility for providing system configuration and tool version information to the Green Hills support staff.
- **Green Hills Probe Administrator (gpadmin)** — A graphical interface for configuring and managing Green Hills Probes and SuperTrace Probes.
- **Graphical Utilities (wgutils)** — A collection of utilities for analyzing and performing various operations on object files, libraries, and executables produced with the Green Hills toolchain.
- **MULTI License Administrator (mlmadmin)** — A graphical utility for managing Green Hills tools licenses.

## The MULTI Launcher

---



The MULTI Launcher (**mstart**) provides a convenient way to launch frequently used tools, to create new or access recently used files and projects, and to manage MULTI workspaces. All of the main MULTI components can be accessed using the following buttons:

-  — Runs a shortcut or an action sequence in the current workspace. Also allows you to create a new workspace or create or edit a shortcut.
-  — Opens the Project Manager on a recent or new project.
-  — Opens the Debugger on a recent or new executable.
-  — Opens the Editor on a recent or new file.
-  — Opens the Checkout Browser on a recent or new checkout.
-  — Establishes a target connection or opens the Connection Chooser or Connection Organizer.
-  — Opens a Serial Terminal using a recent or new connection.
-  — Opens the EventAnalyzer (licensed separately).
-  — Opens the ResourceAnalyzer (licensed separately).
-  — Shows/hides the detail pane of the Launcher.

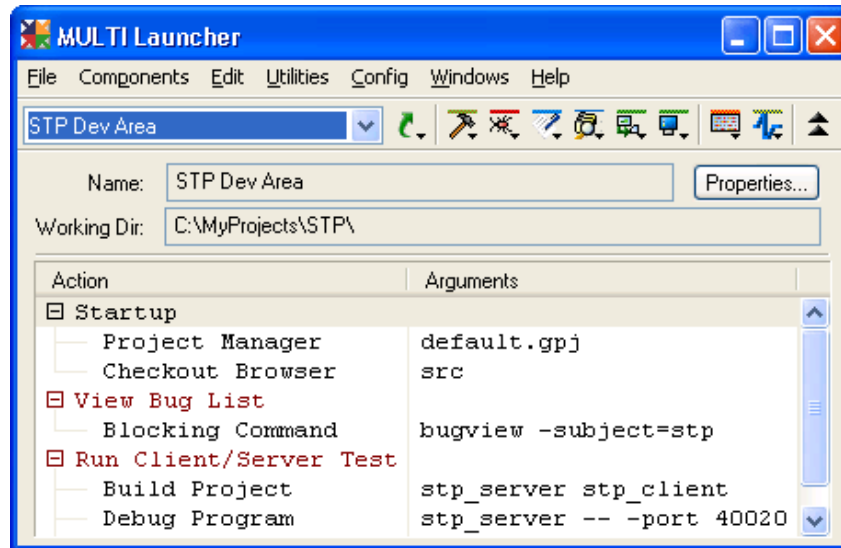
You can also launch the Green Hills License Administrator and (if installed) the Green Hills Probe Administrator from the **Utilities** menu.

During development, you can use the MULTI Launcher as a convenient, centralized window manager. You can access any of your open MULTI windows from the **Windows** menu of the Launcher.



## MULTI Workspaces

The MULTI Launcher allows you to create and use *workspaces*. A MULTI workspace is a virtual area where the tools, files, and actions required for a particular project can be organized, accessed, and executed.



A workspace is typically created for each Top Project, and includes a working directory and a group of related *actions*—for example, opening a project in the MULTI Project Manager, connecting to a target, or performing a shell command. Actions are grouped into action sequences, so that a single mouse click can perform all the actions in the specified action sequence.

For more information, see Chapter 5, “Using MULTI Workspaces and Shortcuts” in the *MULTI: Editing Files and Configuring the IDE* book.

## Accessing Online Help

---

Online help provides useful information about your MULTI tools and can be accessed in several different ways. The following sections describe how to access online help and how to use the Help Viewer.

### Full Online Manuals

You can access an indexed, hypertext version of any MULTI manual by selecting it from the list that appears in the **Help** → **Manuals** menu. The MULTI Help Viewer opens on the manual specified.

You can also access manuals from the command line. Ensure that the MULTI installation directory is in your path and enter the following command:

```
helpview pathname | -m manual_name
```

where:

- *pathname* opens the **.chm** file specified in *pathname*. You must provide a full path.
- -m *manual\_name* opens the manual *manual\_name*. An example manual name is "MULTI: Debugging". If the manual name contains a space as in the preceding example, you must enclose it with quotation marks.

### Context-Sensitive Help

Many MULTI windows and dialog boxes are linked to specific sections of the online manuals. To view the Help Viewer page that documents an active window or dialog box, press F1.

If you are using the MULTI Editor, you can also open context-sensitive help about a button or a menu item by selecting **Help** → **Identify** and then clicking the button or selecting the menu item.

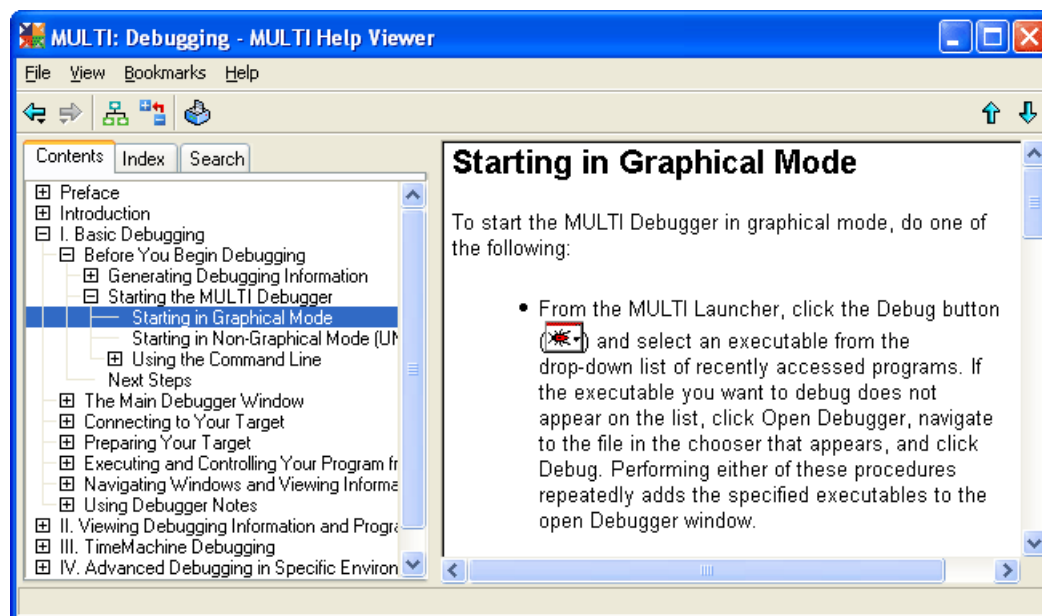
### Command and Configuration Option Help

You can obtain help information about a specific MULTI Debugger command or MULTI configuration option by typing `help command_name` or `help configuration_option` in the Debugger command pane. This opens the Help

Viewer on documentation for the specified command or configuration option. You can also type `usage command_name` to print the basic syntax of a MULTI command to the command pane.

## The Help Viewer Window

The following graphic displays the Help Viewer window.



The Help Viewer toolbar contains the following buttons, from left to right:

- **Back** (←) and **Forward** (→) — Returns to pages you have visited during the current help session. Click the button once for each page you want to revisit. Click and hold the button to select from a list of pages you have visited. These buttons function across books.
- **View All Subtrees** (📁) — Toggles highlighting of the current selection's sub-sections (if any) in the **Contents** tab, and toggles the display of help for the current selection's sub-sections (if any) in the view pane. To disable highlighting in the **Contents** tab and to display help for only the selected item, click this button again. When you open the Help Viewer, this button defaults to its last value.
- **Contract All Unselected** (📁) — Collapses hierarchies located in the **Contents** tab if the hierarchies do not contain any selections.

- **Print** () — Prints the help page displayed in the view pane. To create a help page consisting of mixed topics, hold down the Ctrl key while clicking separate topics. MULTI combines the separate topics and orders them by their location in the **Contents** tab.
- **Previous Page** () and **Next Page** () (located at far right) — Displays the previous or following section as ordered in the **Contents** tab. If the previous or next section includes sub-sections and the **View All Subtrees** button is enabled, the Help Viewer displays the sub-sections (if any) along with the section.

The MULTI Help Viewer contains two panes. The right-hand pane, or the *view pane*, displays the help page for your selection. The left-hand pane contains the following three tabs:

- **Contents** — Displays the parts, chapters, sections, and sub-sections for the manual you are viewing. Click a plus or minus icon to show or hide headings for embedded sections. Click a heading to display the associated help page in the view pane.
- **Index** — Displays index entries for the manual. To search the index entries, enter a word or words in the text box located at the top of the pane. MULTI returns results that contain the word(s) you typed. Click an index entry to display the corresponding help page in the view pane.
- **Search** — Provides an interface that allows you to search for specific words in the current manual or in all manuals.

## Navigating Manuals in the Help Viewer

Navigating manuals in the MULTI Help Viewer is easy. This section describes different methods of finding information.

### Viewing Previous and Next Sections

To display the previous or following section as ordered under the **Contents** tab, do one of the following:

- Click the  or  button located in the upper-right corner of the Help Viewer window.
- Select **View** → **Previous Contents Entry** or **View** → **Next Contents Entry**.

If the previous or next section includes sub-sections and the **View All Subtrees** button () is enabled, the Help Viewer displays the sub-sections (if any) along with the section.

## Returning to Previously Viewed Help Pages

To return to pages you have visited during the current help session, do one of the following:

- Click the  or  button located in the left corner of the toolbar. Click the button once for each page you want to revisit. Click and hold the button to select from a complete list of pages you have visited during the current help session. These buttons function across books.
- Select **View** → **Back** or **View** → **Forward** from the menu bar. This feature functions across books.

## Linking to Related Topics

The underlined links available in the view pane allow you to jump to pages related to the topic you are viewing.

If, when you click a link, a dialog box appears with the message:

You have requested information that the Help Viewer is unable to retrieve.

one of the following factors may be the cause:

- The manual that the link points to may not be included with your distribution. For a list of available manuals, look in the manuals directory (or directories) of your Green Hills Software distribution(s).
- If the link points to an INTEGRITY or u-velOSity manual, you may be using a version of the operating system that does not support links from MULTI help.
- If the link points to an INTEGRITY or u-velOSity manual, you may need to set the location of the installed OS distribution so that the Help Viewer can locate the relevant help files. For information about how to do this, see “Using MULTI with INTEGRITY or u-velOSity” in Chapter 2, “Installation” in the *MULTI: Getting Started* book.



**Note** If you are using MULTI with multiple target architectures (such as ARM and PowerPC), tools of the same version number should be installed into a single directory, regardless of differing architectures. Note, however, that links to target-specific books may yield unpredictable results in this situation.

### Bookmarking Help Pages

You can use bookmarks to return to pages you have visited across MULTI sessions. To bookmark a page, select **Bookmarks** → **Bookmark This Page**.

You can bookmark a help page consisting of mixed topics by holding down the Ctrl key while clicking separate topics. MULTI combines the separate topics in the view pane and orders them by their location under the **Contents** tab. Bookmark the page as usual.

The **Bookmarks** menu lists all your bookmarks from across MULTI manuals and sessions. To return to a bookmarked page, do one of the following:

- Select **Bookmarks** and choose a bookmark.
- Select **Bookmarks** → **Manage Bookmarks**, click a bookmark, and click the **Show** button (.
- From any MULTI tool, select **Help** → **Bookmarks** and choose a bookmark.

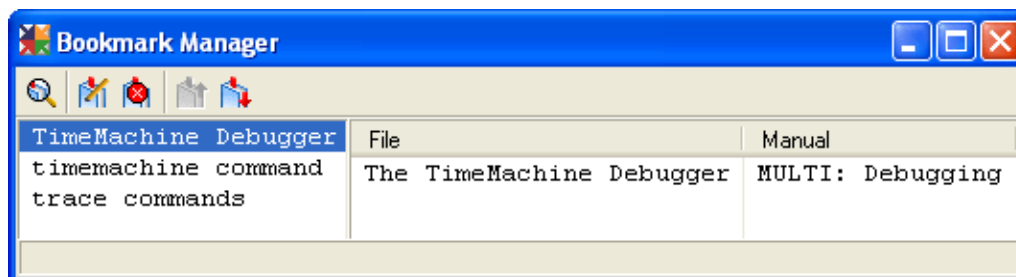
To rename, delete, or reorder existing bookmarks, launch the **Bookmark Manager** as described in the next section, “Managing Bookmarks” on page 10.

### Managing Bookmarks

You can use the **Bookmark Manager** to rename and delete bookmarks, navigate to bookmarks, and modify the ordering of bookmarks in the **Bookmarks** menu.

To open the **Bookmark Manager**, do one of the following:

- In the Help Viewer, select **Bookmarks** → **Manage Bookmarks**.
- From any MULTI tool, select **Help** → **Bookmarks** → **Manage Bookmarks**.



The list on the left side of the **Bookmark Manager** displays the names of all the bookmarks that you have created. When a bookmark is selected, the file(s) and the manual marked by that bookmark are displayed in the right side of the window. The bookmark may reference multiple files (help pages) if, for example, you created the bookmark with multiple entries selected in the **Contents** tab or if you selected an index entry that pertained to multiple files.

The toolbar buttons in the **Bookmark Manager** allow you to perform the following operations on bookmarks:

- — Display the selected bookmark(s) in the Help Viewer.
- — Rename the selected bookmark.
- — Delete the selected bookmark(s).
- and — Move the selected bookmark up or down in the list. The order of the bookmarks in the **Bookmarks** menu mirrors the order of the bookmarks in the **Bookmark Manager**.



**Note** The and buttons are available for use on multiple bookmarks. You can select multiple bookmarks from the left side of the window by dragging to select rows or by using Shift or Ctrl to select items.

## Opening Additional Manuals from the Help Viewer

To open a new manual in the current window, do one of the following:

- Select **File** → **Open Manual** and choose from the list of installed manuals.
- Select **File** → **Open Manual File** and navigate to a particular file in the file chooser that appears.

To open a new manual in a new window, select **View** → **Open Other Manuals in New Window** and then open the manual as usual. When you open the Help

Viewer, the **Open Other Manuals in New Window** toggle option defaults to its last value.



# Setting Up Your Target Hardware

---

## Chapter 2

*This Chapter Contains:*

- Installing Your Target Hardware
- Configuring Your Target Hardware
- Specifying Setup Scripts

This chapter describes how to set up your target hardware for use with MULTI. This chapter is not relevant if you are connecting to a simulated target.

### Installing Your Target Hardware

---

Before you can configure your embedded target for use with MULTI, you must install your hardware and any necessary software. Your installation procedure will vary according to your particular system. For detailed instructions, see your hardware documentation and the chapter in this book that discusses your debug server (see “Supported Target Connections for PowerPC” on page v for a list of supported targets and the relevant chapter for each).



**Note** The Green Hills debug servers documented in this book are installed automatically when you install MULTI. For basic information about installing MULTI, see the *MULTI: Getting Started* book.

### Configuring Your Target Hardware

---

Before beginning your first debugging session, you should ensure that your target board is configured properly. For CAT-DEV setup see documentation included with the Cafe SDK. The next section describes how you can generate the required setup and linker directives files using the **Project Wizard** and Project Manager. If you need a more detailed description of these tools and their capabilities, see Chapter 2, “The MULTI Project Manager” in the *MULTI: Building Applications* book.

### Using the Project Wizard to Configure your Debugging Environment

The **Project Wizard** is a tool that creates a new project structure and generates a board setup script, linker directives files (if applicable), and Connection Methods based on information you provide about your target.

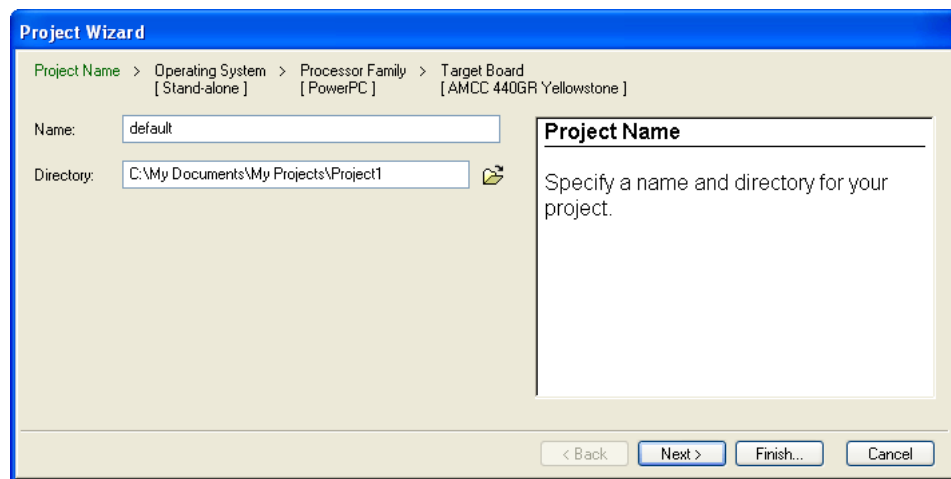
If the wizard created a board setup script for your Top Project, you can run the script, and then use one of the example projects available in the Project Manager to test whether the script configured your target board properly. If you are able to compile, download, and debug a test application from the example project, your debug connection is working correctly and no additional board configuration is required. In most cases, running the script that the wizard creates will configure your target board properly.



**Note** If the **Project Wizard** does not contain an example project or create a board setup script for your target, you might need to create a custom setup script and/or linker directives files. See “Testing and Customizing your Debugging Environment” on page 17 for more information.

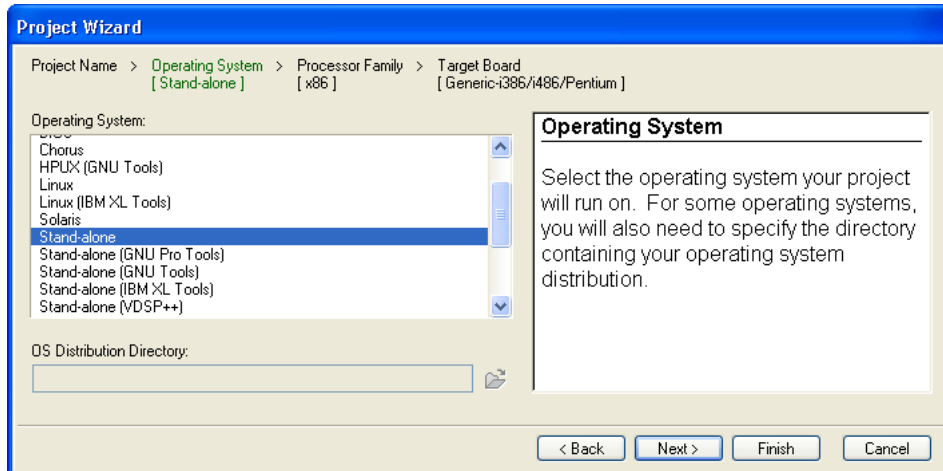
To use the **Project Wizard** and Project Manager to create an example project that you can use to test your target board configuration and connection, perform the following steps:

1. From the MULTI Launcher, click  and select **Create Project** to open the **Project Wizard**.
2. For this example project, do not change the text in the **Name** or **Directory** boxes on the first wizard screen. The **Project Wizard** will create your project with the default name in the default directory.



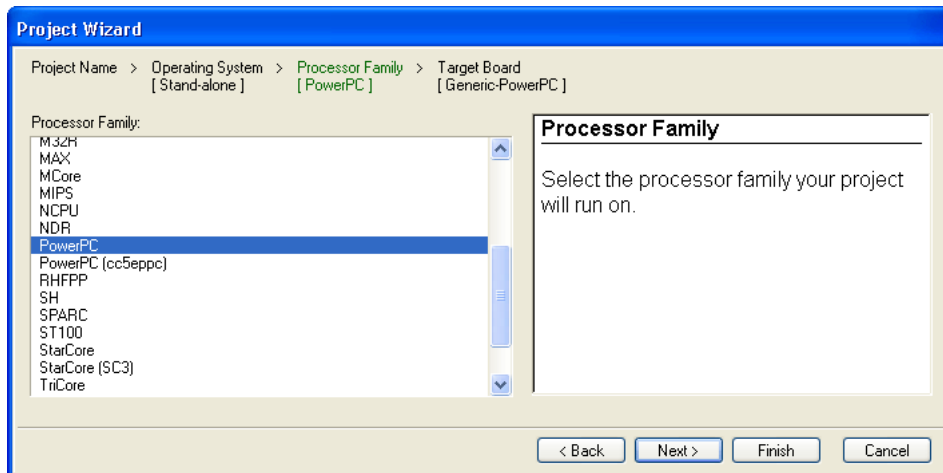
Click the **Next** button.

3. In the **Operating System** list, choose the operating system running on your target.



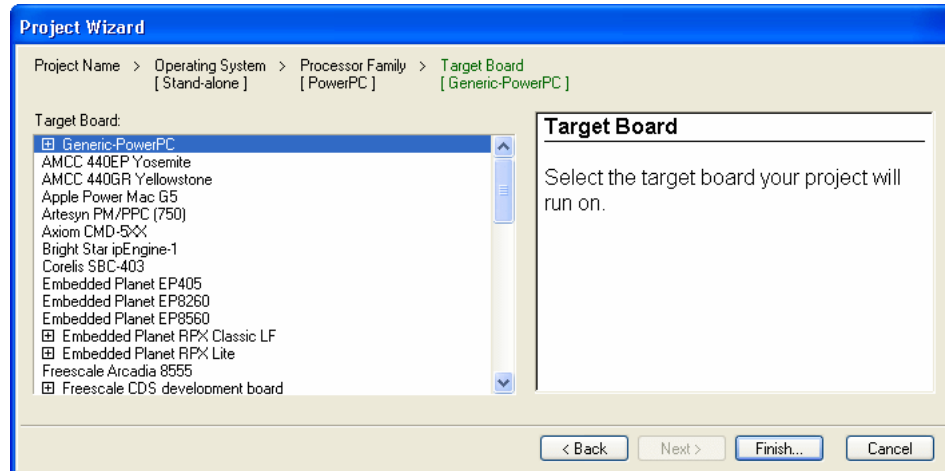
Click the **Next** button.

4. In the **Processor Family** list, choose the processor family to which your target's processor belongs. If you have only installed one processor family for MULTI, the wizard may skip this screen.



Click the **Next** button.

5. In the **Target Board** list, choose your target board. If your target board is not listed, click the plus sign to the left of the **Generic** item at the top of the list to expand it, then choose your target's processor. In this case, you may need to modify some of the files created by the wizard before your example project will run properly.



6. You can click the **Finish** button any time it is available to accept the default options on the remaining screens. For more information about the **Project Wizard** screens and options, see “Creating a New Project” in Chapter 2, “The MULTI Project Manager” in the *MULTI: Building Applications* book for your processor family.

When you click the **Finish** button, the **Project Wizard** closes and a dialog box opens to inform you about the newly created framework Top Project. To disable this message in the future, select the **Never show this message again** check box. Click the **OK** button to continue.

7. After you create a new Top Project, the Project Manager will give you the opportunity to add executables and examples to it (see “Adding Executables and Examples to the Top Project” in Chapter 2, “The MULTI Project Manager” in the *MULTI: Building Applications* book). Choose **Hello World (C)**, then click the **Finish** button. The MULTI Project Manager opens, displaying the files created by the wizard for your project.
8. In the Project Manager, select the **src\_hello\hello.gpj** project and click  to build it.

You can now connect to your target to test whether or not the debugging environment created by the **Project Wizard** configures your target properly.

## Testing and Customizing your Debugging Environment

Once you create a project with the **Project Wizard** (see “Using the Project Wizard to Configure your Debugging Environment” on page 14), test your debugging environment to see if it is working properly before you download

and debug a program. If the debugging environment does not work properly, you might need to customize setup scripts and/or linker directives files in your Top Project. The following sections provide general instructions for testing and customizing your debugging environment:

1. “Testing Your MULTI Board Setup (.mbs) Script” on page 18
2. “Editing a MULTI Board Setup (.mbs) Script” on page 21
3. “Editing Linker Directives (.ld) Files” on page 30

In MULTI 5, the **Project Wizard** creates MULTI board setup (.mbs) scripts. These scripts use the MULTI scripting language described in Chapter 2, “Using MULTI Scripts” in the *MULTI: Scripting* book, and commands described in the *MULTI: Debugging Command Reference* book. The following sections assume that you are using MULTI board setup scripts.

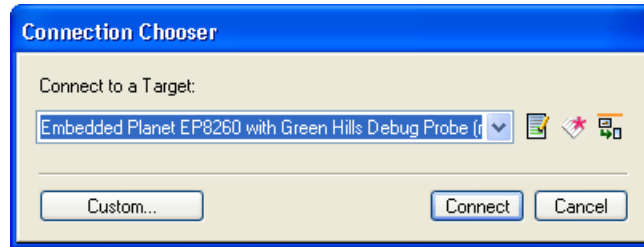


**Note** Board setup scripts in previous versions of MULTI are legacy debug server (.dbs) setup scripts. These scripts use deprecated commands and scripting conventions. If you have developed a project in an earlier version of MULTI and want to use your legacy script with your MULTI 5 installation, the procedure for customizing that script is similar to the one that follows. See Appendix A, “Green Hills Debug Server Command and Scripting Reference” for a command reference and examples that are appropriate for legacy scripts. Also, see “Specifying Setup Scripts” on page 31 for information about how to specify use of these scripts in MULTI 5.

### Testing Your MULTI Board Setup (.mbs) Script

Before you attempt to download and debug a program, test the MULTI board setup (.mbs) script that the **Project Wizard** created for you by following the steps below:

1. Verify that all hardware connections are tight and secure, and that both the board and your hardware interface (if applicable) are powered.
2. Connect MULTI to your target as follows:
  - a. Select your project, and click  in the Project Manager to open the MULTI Debugger.
  - b. In the Debugger window, click , which opens a **Connection Chooser** window.



In the **Connect to a Target** list, select the combination of target and debug server you want to use. In the example shown here, a **Green Hills Debug Probe (mpserv)** connection has been selected.

- c. Click the **Connect** button. If the **Connection Chooser** needs more information, it opens the **Connection Editor**. In this case, set the required options in the **Connection Editor** and click the **OK** button (see “The Connection Editor” in Chapter 4, “Connecting to Your Target” in the *MULTI: Debugging* book for more information).



**Note** When the **Project Wizard** creates a Connection Method, it sets the **Board Setup Script** field to one of the scripts it generated with your Top Project. If your target board requires a board setup script, but you are not using a Connection Method created by the **Project Wizard**, or if you want to use a script other than the default script selected by MULTI, specify a script filename (and pathname, if necessary) in the **Board Setup Script** field. MULTI will run the specified script before each download.

- d. Click **Connect** in the **Connection Chooser**.
3. Run the **setup** command in the MULTI command pane.
  4. To test reading and writing to registers:
    - a. Select a general purpose register (for example, `r1`).
    - b. Read the register and note its value. For example, enter the following in the Debugger command pane:

```
$r1
```

- c. Write a different value to the same register. For example, enter the following in the Debugger command pane:

```
$r1=0xdeadbeef
```

- d. Read the register again and see if it has changed to the new value. If it has, the target connection is able to alter registers.
5. Test whether you can access the target memory through your debugging interface. We recommend that you test memory locations where you are planning on downloading a program. To test the target memory:
  - a. Read a memory location and note its value. For example, enter the following in the Debugger command pane:

```
memread 4 0x20000
```
  - b. Write a different value to the same memory location. For example, enter the following in the Debugger command pane:

```
memwrite 4 0x20000 0xdeadbeef
```
  - c. Read the memory location again and see if it has changed to the new value. If it has, the debugging interface is successfully accessing the target memory.



**Note** You can also use the graphical **Memory Tester** to test your target memory. See Chapter 19, “Testing Target Memory” in the *MULTI: Debugging* book for more information.

If your target configuration does not function correctly, you may need to customize the setup script in your Top Project. See “Editing a MULTI Board Setup (.mbs) Script” on page 21 for more information about how to customize this script.

If your target configuration appears to be functioning correctly, you are ready to download your program and begin debugging. If you have problems downloading, running, or debugging your program on your target, you may need to customize your linker directives (.ld) files (see “Editing Linker Directives (.ld) Files” on page 30). For more information about using the MULTI Debugger, see the *MULTI: Debugging* book.



## Editing a MULTI Board Setup (.mbs) Script

After the **Project Wizard** creates a Top Project, the Project Manager opens and displays your project hierarchy. You can find the files that help MULTI connect to and configure your target in the target resources project (**tgt/resources.gpj**). Expand the target resources project and look for a MULTI board setup (**.mbs**) script file called **xserv\_standard.mbs**, where *xserv* represents the debug server that supports your specific debugging interface, as listed in the table below. For example, the default setup script file for the server that supports Green Hills Debug Probe connections is **mpserv\_standard.mbs**.

| Debugging interface | Debug server name<br>(substitute for <i>xserv</i> ) |
|---------------------|-----------------------------------------------------|
| Cafe OS Debug Agent | <b>cafeserv</b>                                     |

Editing a setup script is a trial-and-error process, so keep the following points in mind:

- Do not download and debug a program on your target until your setup script is fully working. If you debug a program with a target that you have not set up properly, the state of the target might be unpredictable.
- If you have connected MULTI to your target (see Chapter 4, “Connecting to Your Target” in the *MULTI: Debugging* book for full instructions), you can use the MULTI Debugger command pane to confirm the success or failure of each command individually instead of trying to debug an entire setup script.
- Some commands cannot be tested individually because they must be executed within a certain time period in relation to other commands. For example, you may need to disable a watchdog timer seconds after you send a reset command. It is difficult to type in the two commands that disable the watchdog timer before the timer expires. In such cases, put the relevant commands into a small script and run the script from the command pane using the < command.
- You can find an overview of useful board setup script commands in “Useful Commands for MULTI Board Setup (.mbs) Scripts” on page 24.

To edit a MULTI board setup script to make it suitable for your system:

1. From the Project Manager, double-click the MULTI board setup script to be edited (for example, **mpserv\_standard.mbs**) to open it in the MULTI Editor.

2. Determine whether your board can initialize itself, and then proceed as indicated in the following:
  - If your target does not have a valid ROM image that initializes the target upon reset, skip to item number 3.
  - If your target has a valid ROM image that initializes the target upon reset, delete the contents of the MULTI board setup script you are editing. Replace the script with the command sequence shown in the following (or an equivalent command sequence).

```
// Reset and halt the board
reset
// Let the ROM image run
target run
// Give the ROM image 3 seconds to set up the board
wait -time 3000
// Halt the board to get ready for debugging
target halt
```



**Note** You may need to alter the **wait** command, depending on how long your board takes to set itself up.

If you need to initialize your board further, continue to the next step. Otherwise, save the setup script and skip to “Editing Linker Directives (.ld) Files” on page 30.

3. Verify that your setup script begins with a command that resets the target, such as the MULTI Debugger **reset** or debug server **target rst** commands. You must halt any target before beginning any debugging activity, or the state of the target might be unpredictable.
4. Configure your target’s memory controller based on your board’s memory resources.

If your memory controller and memory resources are already properly configured, skip to item number 6. The following are general steps for configuring memory using a setup script:

- a. Determine what memory resources your board has by answering the following:
  - How fast and how big is the board’s memory, and where do you want to map it?
  - Does the board have SRAM? If so, where is it?

- Does the board have DRAM? If so, where is it, and where is the DRAM controller for it?
  - Does the DRAM controller need refresh timing information or knowledge of any special modes the DRAM chips may have, such as Synchronous DRAM?
  - Does the board require a peripheral memory base register to access memory controllers or other on-chip peripherals?
- b. If your processor requires you to set up the base register before you can access your memory controllers or other on-chip peripherals, set the base register.
  - c. Using your processor's documentation and memory resources, determine which memory-related registers you must set. Additionally, determine what values those registers must have to properly configure your memory resources.
  - d. Copy the commands necessary to configure your memory resources into your setup script (see "Useful Commands for MULTI Board Setup (.mbs) Scripts" on page 24 for more information).
5. Disable any interrupt sources that can disrupt the setup or destabilize the board's memory. (For example, if you are using a PowerPC 860 processor, disable the watchdog timer to prevent it from interrupting your target board setup.) The following steps provide a general procedure for disabling interrupt sources:
- a. Determine whether your processor has any interrupt sources that might disturb your debugging session.
  - b. Using your processor's documentation, determine which registers affect interrupt sources. Then, determine the values those registers must have to disable the interrupt sources.
  - c. Copy the commands necessary to configure your memory resources into your setup script (see "Useful Commands for MULTI Board Setup (.mbs) Scripts" on page 24 for more information).
6. Verify that your setup script contains all of the commands required to prepare the target system, and then save it.

### Useful Commands for MULTI Board Setup (.mbs) Scripts

Scripting conventions for MULTI board setup (.mbs) scripts are described in Chapter 2, “Using MULTI Scripts” in the *MULTI: Scripting* book, and you can find a complete list of MULTI commands in the *MULTI: Debugging Command Reference* book. However, only a small subset of these commands are useful for most setup scripts. The list below outlines these commands.

|                                                                                                                                                                                                                                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>memread</b>                                                                                                                                                                                                                                                                                                       |
| Performs a sized memory read from the target, and prints the result.                                                                                                                                                                                                                                                 |
| <b>memwrite</b>                                                                                                                                                                                                                                                                                                      |
| Performs a sized memory write to the target.                                                                                                                                                                                                                                                                         |
| <b>reset</b>                                                                                                                                                                                                                                                                                                         |
| Resets the target.                                                                                                                                                                                                                                                                                                   |
| <b>wait</b>                                                                                                                                                                                                                                                                                                          |
| Blocks command processing.                                                                                                                                                                                                                                                                                           |
| <code>\$register = value</code>                                                                                                                                                                                                                                                                                      |
| Sets a register named <i>register</i> on your target board to <i>value</i> . For example, if you put <code>\$ivor0 = 0x10</code> in your script, MULTI sets the register named <code>ivor0</code> to <code>0x10</code> . You can find information about your target board’s registers in your target board’s manual. |

## Example Scripts

This section includes example MULTI board setup scripts.

### Example 1. MULTI (.mbs) Board Setup Script

```
// This is an example MULTI Board Setup Script which sets up the
// Motorola MBX for running programs. The board is configured to
// run at 40 MHz.

// delay slow
// Reset the processor
target rst
// set the internal register space to 0xff000000
$immr=0xff000000
// turn off the wchdog timer --- disable always
memwrite 4 0xff000004 0xffffffff88
// disable the cache
$dc_cst=0x04000000
// The board is configured to run at 40 MHz
// init set_clock=1 cpu_speed=40000000
// delay fast

// Setup memory
memwrite 4 0xFF00017C 0xCFAFC004
memwrite 4 0xFF000168 0x00000000

...

memwrite 2 0xFF00017A 0x0200
```

### Example 2. MULTI Board Setup (.mbs) Script for an MPC55xx Board

This example script prepares a Freescale MPC55xx board for download. The variables at the beginning of the script allow you to configure its operation.

```
// MULTI Server Script
// Freescale MPC55xx demo board setup script for MPServ
// This script has been tested on all 55xx board models
// available as of December, 2006, including models with
// MPC5534, MPC5553, MPC5554, MPC5561, MPC5565, MPC5566, and MPC5567 cpus.
// It has not been tested with any 551x processors and may not work
// with those models without additional modification.

// reset
target rst
target rw msr 0x02000000      // set MSR[SPE]

// Whether or not to check for VLE
$vle_check = 1
if ($PROGRAMMING_FLASH) {
    $vle_check = 0
}

// Whether or not to check for 16-bit port size on external SRAM
$bus_check = 1

// Whether or not to enable cache
$cache_enable = 0

// Which chip select to use for external SRAM
// This is determined by the SRAM_SEL jumper
// 1-2 closed: CS0   (default)
// 2-3 closed: CS1
$chip_select = 0

// The starting address for the external 512K SRAM
// Starting at 0x3ff80000 makes it adjacent to the internal 64K, making
// for one contiguous 576K block.
$baseAddr = 0x3ff80000
substitute target bAddr = %EVAL{$baseAddr}

// Initialize vector pointers to something that exists
// Currently, use internal SRAM
$ivpr = 0x40000000
$ivor0 = 0x10
$ivor1 = 0x20
$ivor2 = 0x30

...

$ivor34 = 0x130
```

```
// Fill exception vectors with branch to self
target mf 0x40000000i 0x200 0x48000000

// tlbw vtag meanings of bits used below:
// V    = 0x01000000
// Size = 0x00780000 (bits 19..22 where bit 0 is lsb)
// I    = 0x00000200
// G    = 0x00000080
// UX   = 0x00000020
// SX   = 0x00000010
// UW   = 0x00000008
// SW   = 0x00000004
// UR   = 0x00000002
// SR   = 0x00000001
//
// tlbw ptag meaning of bit used below:
// Iprot = 0x1

// Setup the TLB to map the internal 64K SRAM
target tlbw 3 0x40000000 0x120003f 0x40000000 0x1

// Setup the TLB to map the external 512K SRAM
target tlbw 4 0x$$bAddr 0x128003f 0x$$bAddr 0x1

// Setup the TLB to map the 1MB needed for Bridge B peripherals
// Set the Guard (G) flag because this entry points to I/O registers.
target tlbw 0 0xfff00000 0x0128028f 0xfff00000 0x1

// Initializes the external bus
// Configure external SRAM

// Enable flash configuration registers
target tlbw 5 0xc3f80000 0x120003f 0xc3f80000 0x1

// Set up the pins
// Address bus PCR 4 - 27 43
// configure address bus pins
memwrite 4 0xc3f90048 0x04400440
memwrite 4 0xc3f9004c 0x04400440
memwrite 4 0xc3f90050 0x04400440
memwrite 4 0xc3f90054 0x04400440
memwrite 4 0xc3f90058 0x04400440
memwrite 4 0xc3f9005c 0x04400440
memwrite 4 0xc3f90060 0x04400440
memwrite 4 0xc3f90064 0x04400440
memwrite 4 0xc3f90068 0x04400440
memwrite 4 0xc3f9006c 0x04400440
memwrite 4 0xc3f90070 0x04400440
memwrite 4 0xc3f90074 0x04400440
```

```
// Data bus PCR 28-59
// configure data bus pins
memwrite 4 0xc3f90078 0x04400440
memwrite 4 0xc3f9007c 0x04400440
memwrite 4 0xc3f90080 0x04400440
memwrite 4 0xc3f90084 0x04400440
memwrite 4 0xc3f90088 0x04400440
memwrite 4 0xc3f9008c 0x04400440
memwrite 4 0xc3f90090 0x04400440
memwrite 4 0xc3f90094 0x04400440

// These next 8 are not required for 16-bit address bus.
memwrite 4 0xc3f90098 0x04400440
memwrite 4 0xc3f9009c 0x04400440
memwrite 4 0xc3f900a0 0x04400440
memwrite 4 0xc3f900a4 0x04400440
memwrite 4 0xc3f900a8 0x04400440
memwrite 4 0xc3f900ac 0x04400440
memwrite 4 0xc3f900b0 0x04400440
memwrite 4 0xc3f900b4 0x04400440

// config minimum bus control pins
// RD/WR & BDIP PCR 62/63
memwrite 4 0xc3f900bc 0x04400440

// WE[0-4] PCR 64-67
memwrite 4 0xc3f900c0 0x04430443
memwrite 4 0xc3f900c4 0x04430443

// OE & TS
memwrite 4 0xc3f900c8 0x04430443

// configure the chip selects
// CS[0-3]
memwrite 4 0xc3f90040 0x04430443
memwrite 4 0xc3f90044 0x04430443

// Set up Memory Controller CS0 or CS1 for external SRAM
// See comments at top of file for which chip select to use
if ($chip_select == 0) {
    memwrite 4 0xc3f84010 ($baseAddr+0x41)
    memwrite 4 0xc3f84014 0xffff800F0
} else {
    memwrite 4 0xc3f84018 ($baseAddr+0x41)
    memwrite 4 0xc3f8401c 0xffff800F0
}

// CLKOUT
//PCR 229
memwrite 2 0xc3f9020a 0x02c0

// Enable internal flash memory
target tlbw 1 0x00000000 0x138003f 0x00000000 0x1
```



```
// Check for 16-bit port size.
// The MPC5553 demo board has ethernet, and the ethernet controller uses
// D16 to D31, leaving D0 to D16 for the external SRAM. So for the external
// SRAM to operate correctly, it needs to be configured with a 16-bit port
// size. We determine if this is needed by writing to memory and reading
// the value back.
// Burst mode only works for 32bit buses, so it needs to be inhibited
if ($bus_check == 1) {
    memwrite 4 ($baseAddr+0x20) 0xdeadbeef
    $memData = *(int*)($baseAddr+0x20)
    // Check that first half of word matches, but second half does not.
    if (($memData != 0xdeadbeef) && (($memData & 0xffff0000) == 0xdead0000))
    {
        print "Configuring External SRAM with 16-bit port size."

        // Set the PS bit in EBI_BR0
        memwrite 4 0xc3f84010 ($baseAddr+0x843)
    }
}

// The next few lines enable the unified L1 cache.
if ($cache_enable == 1) {
    $llcsr0 = 0x3
}

// Check for VLE.
// If the .vletext exists, we will assume that the entire program is built
// in VLE mode, and set both internal and external SRAMs accordingly.
// If different/additional pages need to have VLE enabled, this is where
// to do it.
if ($vle_check == 1) {
    if ($M_sec_exists(".vletext")) {

        print "Configuring TLB pages for VLE."

        // Turn on VLE bit for internal SRAM
        target tlbw 3 0x40000000 0x320003f 0x40000000 0x01

        // Fill exception vectors with VLE branch to self
        target mf 0x40000000 0x200 0xe800e800

        // Turn on VLE bit for external SRAM
        target tlbw 4 0x$$bAddr 0x328003f 0x$$bAddr 0x1
    }
}

// Set CPU to run at 128 MHz
// FMPLL_SYNCR = 0x06000000
// MFD = 0x6, RFD = 0x0 -> 16x external 8 MHz clock -> 128 MHz
memwrite 4 0xc3f80000 0x06000000
```

```
// Check for e200z3 core; on such processors, enable crossbar arbitration
// so that the probe can read memory while the target is running.
// e200z3 has no cache, that's how we'll check if this is an e200z3.
$regData = $L1CFG0
if ($regData == 0) {

    // Turn on round robin arbitration for EBI memory slave
    // XBAR_SGPCR1 = 0x100
    memwrite 4 0xffff04110 0x100

    // If necessary, set other slaves (such as flash) for round robin
}
```

### Editing Linker Directives (.ld) Files

A linker directives (.ld) file controls how the linker links your executable and loads it into memory. When you create a project for a target running Stand-Alone programs using the **Project Wizard**, the wizard will create several linker directives files and place them in the Top Project's target resources project (**tgt/resources.gpj**). The linker will link any project you add to your Top Project using one of these files, depending on that project's **Program Layout** setting.

By default, the **Project Wizard** sets the **Program Layout** setting for new Stand-Alone program projects to **Link to and Execute out of RAM**. Because of this setting, the linker will link the program using the **standalone\_ram.ld** linker directives file in your target resources project. You will also see a duplicate entry for this file as **tgt/standalone\_ram.ld** inside the new program project. If, in the **Project Wizard**, you chose a board that has a different memory map than your target board, you must edit this file.



**Note** You can change the **Program Layout** for your project by right-clicking the project and choosing **Configure**. If you change the **Program Layout**, you will need to edit a different linker directives file. See “Creating a New Project” in Chapter 2, “The MULTI Project Manager” in the *MULTI: Building Applications* book for more information.

This section provides the general procedure for editing linker directives files. For more information about linker directives files, see the *MULTI: Building Applications* book for your processor family.

1. In the Project Manager, expand the target resources project.

2. Double-click the linker directives (**.ld**) file you want to edit in the MULTI Editor and edit the file as necessary.
3. Save the edited linker directives file.
4. Rebuild your project by selecting the project (**.gpj**) file in the Project Manager and clicking .

## Specifying Setup Scripts

---

After you have used the **Project Wizard** to generate a setup script for your target or have created a customized setup script based on a default script created by the wizard, you should run the script prior to every download to ensure that your target is clean, stable, and properly configured. The procedure for specifying and running target setup scripts varies slightly depending on how you are connecting to your target, and what type of setup script file you are using.

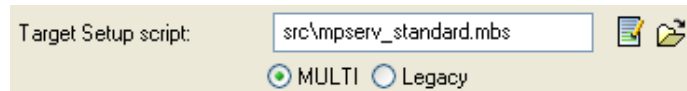
Target setup script files in previous versions of MULTI had **.dbs** extensions. Appendix A, “Green Hills Debug Server Command and Scripting Reference”, demonstrates the debug server commands and scripting language used in these files. Beginning with MULTI 5.0, the debug server scripting language has been deprecated. Setup script files now have an **.mbs** extension and use MULTI commands and scripting conventions. The MULTI scripting language and conventions are described in Chapter 2, “Using MULTI Scripts” in the *MULTI: Scripting* book. MULTI 5 supports both **.dbs** and **.mbs** script files. Therefore, when you specify a setup script you must also specify which scripting language the script uses, so that MULTI interprets the script correctly. You can do this from the **Connection Editor** by selecting the appropriate scripting language radio button (see “Using MULTI (.mbs) Setup Scripts When Connecting to Your Target” on page 32).

If you are connecting using a command line instruction or a Custom Connection Method, the syntax for specifying a **.dbs** script is slightly different than that for specifying an **.mbs** script. The subsequent sections summarize the various ways to specify and run setup scripts. For detailed instructions on locating, creating, or editing a setup script file for your particular system, see “Using the Project Wizard to Configure your Debugging Environment” on page 14.

## Using MULTI (.mbs) Setup Scripts When Connecting to Your Target

The method for specifying an **.mbs** setup script at the time of connection varies depending upon the procedure you use to connect MULTI to your target. The various methods are:

- To run an **.mbs** setup script every time you connect using a particular Standard Connection Method, specify the filename of the script in the **Target Setup Script** field of the **Connection Editor** for the Connection Method and select the **MULTI** radio button immediately below the field.



If you are using a default Connection Method created by the **Project Wizard**, the necessary setup script file for your processor-board combination (if applicable) is specified automatically and the **MULTI** button will be selected.

- To run an **.mbs** setup script and connect to your target using a Custom Connection Method:
  - If you are editing the Custom Connection Method using the **Connection Editor**, specify the filename of the target setup script in the **Target Setup Script** field.
  - If you are entering the connection command using the **Start a Custom Connection** field of the **Connection Chooser**, precede the debug server command with the command **setup=filename** (where *filename* is the **.mbs** setup script filename). Click **Connect** to continue.
- To run an **.mbs** setup script from the Debugger command pane, use the following syntax:

**connect setup=filename.mbs dbserv [args]... [opts]...**

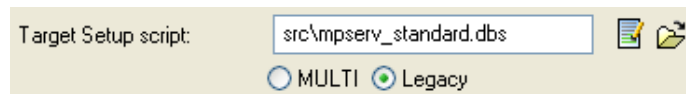
where *filename.mbs* is the setup script filename, *dbserv* is the name of the debug server to be used, and *args* and *opts* are appropriate arguments for your debug server and target.

For more information about the **connect** command, see “General Target Connection Commands” in Chapter 18, “Target Connection Command Reference” in the *MULTI: Debugging Command Reference* book and Chapter 4, “Connecting to Your Target” in the *MULTI: Debugging* book.

## Using Legacy Debug Server (.dbs) Setup Scripts When Connecting to Your Target

The method for specifying a **.dbs** setup script at the time of connection varies depending upon the procedure you use to connect MULTI to your target. The various methods are:

- To run a **.dbs** setup script every time you connect using a particular Standard Connection Method, specify the filename of the target setup script in the **Target Setup Script** field of the **Connection Editor** for the Connection Method. Select the **Legacy** radio button immediately below the field.



- To run a **.dbs** setup script and connect to your target using a Custom Connection Method, include the **-setup filename.dbs** debug server option in the command you enter into the **Start a Custom Connection** field of the **Connection Chooser** and then click **Connect**. The debug server option for specifying **.dbs** setup scripts must proceed the debug server command. See the appropriate debug server chapter later in this book for more information about connecting to your specific target this way.
- To run a **.dbs** setup script from the Debugger command pane, use the following syntax:

```
connect dbserv -setup filename.dbs [args]... [opts]...
```

where *filename.dbs* is the setup script filename, *dbserv* is the name of the debug server to be used, and *args* and *opts* are appropriate arguments for your debug server and target.

## Running Setup Scripts Manually

In addition to running setup scripts as part of the connecting process, you can also run setup scripts manually at other times using any of the following methods:

- (MULTI **.mbs** scripts) Run your setup script file manually from the Debugger command pane using the **<** command.
- (MULTI **.mbs** scripts) Use the MULTI **setup filename.mbs** command. If this command is used with a connection command, the setup script runs prior to

downloading and debugging. The **setup** command can also be used without a specific script name if you are connected and specified a setup script when you connected. The script you specified for the connection is ran if you issue the **setup** command with no specified file. See *MULTI: Debugging Command Reference* for more information about the **setup** command.

- (Legacy **.dbs** scripts) Use the **target script filename.dbs** command from the Debugger command pane.

### Early MULTI Board Setup Scripts with Debugger Hooks

Ordinarily, **.mbs** scripts are run every time you download a program to your target, just before the download begins. However, in certain circumstances and for certain targets (such as multi-core boards), it is not appropriate to re-initialize the entire board every time you download a program to a given CPU on that board.

If you set up an **.mbs** script with a comment on the first line containing the marker `MBS_OPT="early"`, the entire board setup script will be run once immediately after you connect to your target instead of every time just before you download a program. The comment should look similar to the following:

```
// MBS_OPT="early"
```

When combined with the **Hook Commands** described in Chapter 15, “Scripting Command Reference” in the *MULTI: Debugging Command Reference* book, the “early” MULTI board setup script mechanism gives you fine-grained control over how and when MULTI will set up your target. For example, you can install hooks in your **.mbs** script to reinitialize the entire board upon reset by using reset hooks to cause certain initializations to take place before reset and certain others to take place afterwards.

```
addhook -before reset { /* Before reset work */ }
addhook -after reset -core 0 { /* After reset, core 0 work */ }
addhook -after reset -core 1 { /* After reset, core 1 work */ }
```

You can also add a hook to reset the board upon connecting, which causes your reset hooks to run whenever you connect to the target with MULTI.

```
addhook -after connect { reset }
```

Lastly, you might decide to reinitialize a selected subset of the board circuitry every time you download a program to one of the CPUs, while leaving the other CPUs alone.

```
addhook -before download -core 0 { /* Core 0's initialization commands */ }  
addhook -before download -core 1 { /* Core 1's initialization commands */ }
```





# Cafe OS (cafeserv) Connections

---

## Chapter 3

*This Chapter Contains:*

- Connecting to Your Target Running Cafe OS

The **cafeserv** debug server allows the user to debug applications built for Cafe OS using MULTI. This chapter supplements the documentation included with the Cafe SDK. Refer to the Cafe Quick Start Guide in the SDK for installation information.

This chapter supplements the general target connection information contained in Chapter 2, “Setting Up Your Target Hardware” of this book and Chapter 4, “Connecting to Your Target” in the *MULTI: Debugging* book with specific information for **cafeserv** connections.

## Connecting to Your Target Running Cafe OS

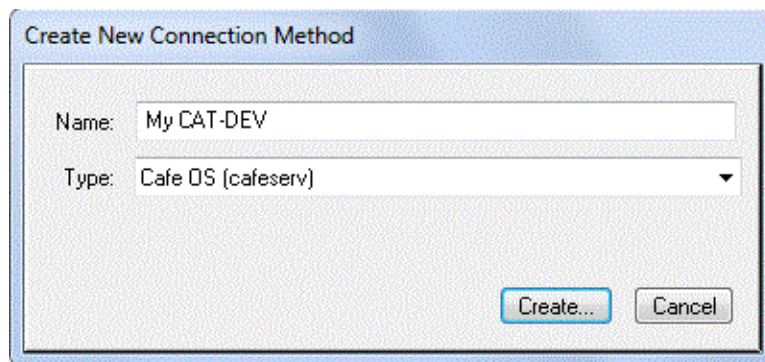
---

You may invoke MULTI using **caferun** (see Cafe Quick Start Guide) or you may connect using the MULTI Connection Organizer and Editor, (see Chapter 2, “Setting Up Your Target Hardware”.) In addition to the generic fields that appear on all Connection Editors for Standard Connection Methods, the **Cafe OS (cafeserv) Connection Editor** includes fields that provide settings and options specific to your target.

For general instructions that explain how to create and use Connection Methods, see Chapter 4, “Connecting to Your Target” in the *MULTI: Debugging* book. The information in the following sections supplements the instructions provided there with information that is specific to **cafeserv** connections.

## Creating a Standard Cafe OS (cafeserv) Connection Method

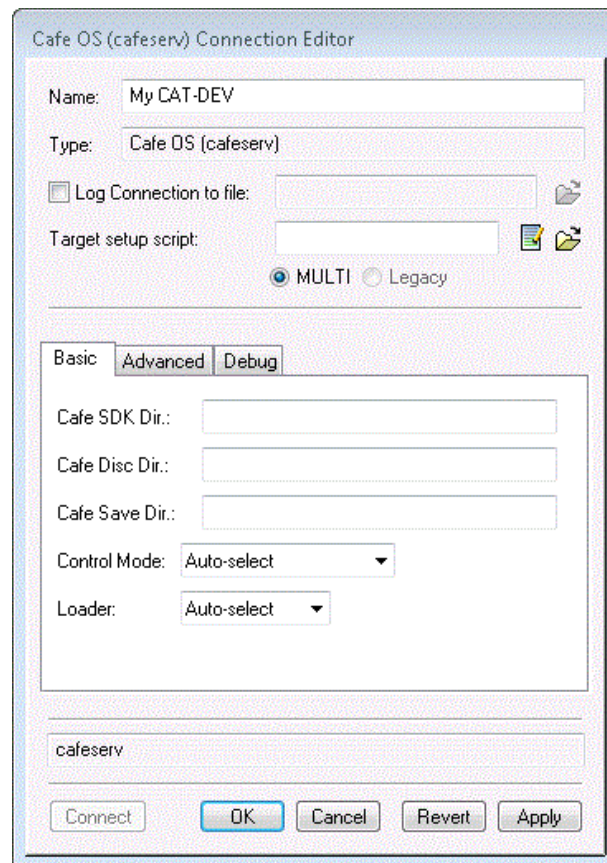
When creating a new Standard Cafe OS Connection Method, select **Cafe OS (cafeserv)** as the connection type in the **Create New Connection Method** dialog box.



For detailed instructions on creating new Standard Connection Methods, see “Standard Connection Methods” in Chapter 4, “Connecting to Your Target” in the *MULTI: Debugging* book.

## Using the Cafe OS (cafeserv) Connection Editor

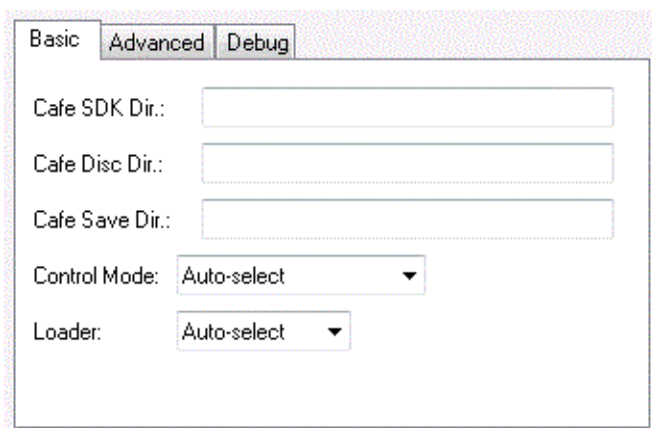
In addition to the generic fields that appear on all Connection Editors for Standard Connection Methods, the **Cafe OS (cafeserv) Connection Editor** includes **Basic**, **Advanced**, and **Debug** tabs that provide settings and options specific to your target operating system.



When the **Connection Editor** is first displayed after you create a new Connection Method, the settings and options are set to default values. Settings and options that are not available on your host operating system may appear dimmed. Some of the fields may require user input before the Connection Method can be used.

All of the fields on the **Basic**, **Advanced**, and **Debug** tabs of the **Cafe OS (cafeserv) Connection Editor** are described in detail below. (See “The Connection Editor” in Chapter 4, “Connecting to Your Target” in the *MULTI: Debugging* book for a description of the other fields and options, which appear on all **Connection Editors**.)

#### Cafe OS (cafeserv) Basic Settings



The screenshot shows a dialog box titled "Cafe OS (cafeserv) Basic Settings". It has three tabs: "Basic", "Advanced", and "Debug". The "Basic" tab is selected. Inside the dialog, there are five fields:

- Cafe SDK Dir.: A text input field.
- Cafe Disc Dir.: A text input field.
- Cafe Save Dir.: A text input field.
- Control Mode: A dropdown menu with "Auto-select" selected.
- Loader: A dropdown menu with "Auto-select" selected.

|                                                           |
|-----------------------------------------------------------|
| <b>Cafe SDK Dir</b>                                       |
| Specifies the location where Cafe SDK has been installed. |
| <b>Cafe Disc Dir</b>                                      |
| Specifies the location where disc content is located.     |
| <b>Cafe Save Dir</b>                                      |
| Specifies the location where save content is located.     |

**Control Mode**

Specifies the method for connecting to and controlling Cafe OS.

- **Auto-select** — [default] MULTI automatically selects whether to connect to the Cafe HostBridge software or to the MION bridge based on the current Cafe SDK and environment. This is recommended.
- **HostBridge (legacy)** — In this mode, **cafeserv** connects to the Cafe HostBridge software on the local host, which routes the communication to Cafe OS on the CAT-DEV.
- **MION Bridge (fast)** — In this mode, which requires Cafe SDK 2.10 or later, **cafeserv** connects to the MION bridge on the CAT-DEV directly. This will be faster than routing communication through the HostBridge software.

**Loader**

Specifies the method for downloading a program or restarting a program within MULTI.

- **Auto-select** — [default] MULTI automatically selects whether to use **caferun** or **CafeX** based on the current Cafe SDK and environment. This is recommended. However, with some Cafe SDK versions prior to Cafe SDK 2.10, which are no longer supported, including some versions of Cafe SDK 2.08 and Cafe SDK 2.09, **CafeX** is incompatible with MULTI and **caferun** must be selected.
- **caferun** — MULTI invokes **caferun** from the Cafe SDK to load the program onto the CAT-DEV. Note that **caferun** may still itself invoke **CafeX** depending on its settings.
- **CafeX** — MULTI invokes **CafeX** from the Cafe SDK to load the program onto the CAT-DEV. This may not work with Cafe SDK 2.08 and Cafe SDK 2.09.

**Cafe OS (cafeserv) Advanced Settings**

The screenshot shows a dialog box titled "Cafe OS (cafeserv) Advanced Settings". It has three tabs: "Basic", "Advanced" (which is selected), and "Debug". The "Advanced" tab contains the following settings:

- Control Hostname:** An empty text input field.
- Control Port:** An empty text input field.
- Disable Output Port:** A checkbox that is currently unchecked.
- Output Hostname:** An empty text input field.
- Output Port:** An empty text input field.
- Caferun Options:** An empty text input field.



**Warning** Use this tab carefully, since changing the advanced options from their default settings can cause problems with your connection.

**Control Hostname**

Specifies the host name or IP address to debug and control the CAT-DEV.

When **Control Mode** is set to **HostBridge**, this must be the host with the Cafe HostBridge software installed, typically the local host. Left blank it defaults to `localhost` in this mode.

When **Control Mode** is set to **MION Bridge**, this must be the name or IP address of the CAT-DEV's MION Bridge. Left blank in this mode, it defaults to the `BRIDGE_CURRENT_IP_ADDRESS` environment variable, if set, otherwise to the IP address of the default bridge configured with the Cafe HostBridge software, if configured, otherwise it is an error.

When **Control Mode** is set to **Auto-select**, this setting will behave according to the automatically selected control mode.

**Control Port**

Specifies the port number to debug and control the CAT-DEV.

Left blank when **Control Mode** is set to **HostBridge**, the multiple-devkit session environment variables are used to select the correct port if available, otherwise it defaults to `6002`.

Left blank when **Control Mode** is set to **MION Bridge**, it defaults to `7977`.

When **Control Mode** is set to **Auto-select**, this setting will behave according to the automatically selected control mode.

**Disable Output Port**

Disables connecting to the debug output port. (This may not disable output, because the SDK may be configured to send output through the control port connection, and indeed this is the default in SDK 2.08.01 and earlier.)

**Output Hostname**

Specifies the host name or IP address for debug output from the CAT-DEV via the Cafe HostBridge software (such as from **OSReport**), which is printed as I/O output in MULTI. Note that debug output is transferred via the Cafe HostBridge software regardless of the setting of **Control Mode**. Left blank it defaults to `localhost`.

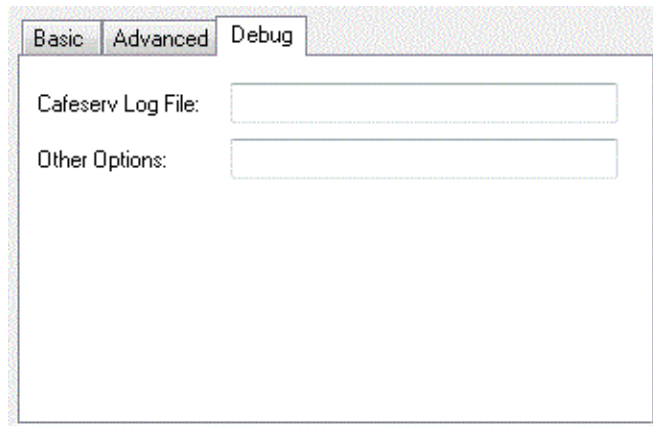
**Output Port**

Specifies the port number for debug output from the CAT-DEV via the Host Bridge software. Left blank, the multiple-devkit session environment variables are used to select the correct port if available, otherwise it defaults to `6001`.

**Caferun Options**

Specifies additional arguments to pass to **caferun/CafeX**.

## Cafe OS (cafeserv) Debug Settings



The screenshot shows a dialog box titled 'Cafe OS (cafeserv) Debug Settings'. It has three tabs: 'Basic', 'Advanced', and 'Debug'. The 'Debug' tab is currently selected. Inside the dialog, there are two text input fields. The first is labeled 'Cafeserv Log File:' and the second is labeled 'Other Options:'.



**Warning** Do not change the settings on the debug tab unless you are instructed to do so by Green Hills Technical Support.

### **Cafeserv Log File**

Specifies a location to output the **cafeserv** debug log, which can be used by Technical Support. It appends to the given file.

### **Other Options**

Allows you to add other, optional arguments directly to the **cafeserv** command line. You should only use this field if directed to do so by Green Hills Technical Support.





# Green Hills Debug Server Command and Scripting Reference

---

## Appendix A

*This Appendix Contains:*

- Green Hills Debug Server Commands
- The Green Hills Debug Server Scripting Language

This chapter describes generic debug server commands and a full scripting language that can be used with most Green Hills debug servers.

These commands and the scripting language are used only in legacy **.dbs** scripts, and are deprecated.

## **Green Hills Debug Server Commands**

---

The commands described below are available to most Green Hills debug servers, but do not apply to simulator or operating system connections.

Most Green Hills debug servers also support additional commands. See “Additional Debug Server Commands” on page 52 for a listing of where to find descriptions of the additional commands (if any) available for the debug server that supports your specific target.

### **Generic Debug Server Commands**

Unless otherwise noted, the commands listed in this section are available for all of the debug servers in this book except **rtserv** and the Simulator for PowerPC (**simppc**), and **cafeserv**.

You can enter all of these commands directly into the debug server **Target** pane. You can also enter these commands into the MULTI Debugger command pane using the **target** command. All of the Green Hills debug server commands are case-insensitive.

**addressof** *symbol*

Returns the address of *symbol*. This command requires that an image be loaded in the MULTI Debugger.

**amask** [*mask*] [*value*]

If no arguments are specified, the current settings of the download mask and value are displayed. The default settings of *mask* and *value* are 0xffffffff and 0, respectively.

If *mask* and *value* are specified, **amask** sets the download mask and value to *mask* and *value*. When the debug server downloads a program, it will bitwise AND *mask* and bitwise OR *value* with the download addresses, as follows:

$$address = (address \& mask) | value$$

The **amask** command is useful for shifting download target addresses without relinking a program. However, the program being downloaded still retains its original relocations and might not run correctly at its new destination address without further support on the target.

**close** *fd*

Closes the specified file descriptor, *fd*.

**debug** *n*

Sets the debugging bit flags.

Do use this command unless directed to do so by Green Hills Technical Support.

**delrange** *addr*

Deletes a checked address range starting at *addr*. See **setrange** for more information about checked memory ranges.

**delrangeall**

Deletes all checked address ranges. See **setrange** for more information about checked memory ranges.

**echo** [on | off]

If no argument is specified, the current echo mode is printed.

If **on** or **off** is specified, printing of commands executed in a script are enabled or disabled.

**fprint** *fd string*

Prints *string* to the specified file, *fd*, with script variable expansion.

**fprintb** *fd integer*

Prints *integer* to the specified file, *fd*, in binary mode.

**fread** *fd identifier*

Reads one line of text from the specified file, *fd*. The line is stored as a variable with the specified *identifier*. The number of bytes read is returned. If there is an error, -1 is returned.

**freadb** *fd identifier*

Reads an integer from the specified file, *fd*, in binary mode. The integer is stored as a variable with the specified *identifier*. The number of bytes read is returned. If there is an error, -1 is returned.

**getenv** *envName identifier*

Stores the value of the environment variable, *envName*, in the script variable with the specified identifier.

**halt**

Halts execution of the target CPU and forces it into debugging mode.

**help** [*command* | *group*]

If no argument is specified, general help information and instructions for finding more detailed and specific help is printed.

If a *command* is specified, detailed help information about the specified *command* is printed.

If a *group* is specified, a list of all of the commands in the group with information about the arguments each command takes is printed. Valid command groups include **target**, **server**, and **scripting**.

Example 1:

```
> help server
help [<command> | <group>]
debug <n>
playdead
```

Example 2:

```
> help help
help [<command> | <group>]
Prints help information
```

**litrage**

Lists all checked ranges. See **setrange** for more information about checked memory ranges.

**listvars**

Prints all variable identifiers in no particular order.

Example:

```
> str1="foo"
> str2="bar"
> i=100
> listvars
i
str1
str2
```

**load [all] [text] [data] [bss]**

If no argument is specified: Displays the current **load** settings.

If one or more arguments are specified, **load** instructs which sections to include in host-to-target downloads. `.text` sections contain code, `.data` sections contain initialized variables, and `.bss` sections contain uninitialized data. Setting the **all** option includes all of these sections in the download.

You can combine the **load** command with the **noload** command.

Green Hills startup code clears all `.bss` sections so you do not need to download them. In most cases, the default setting is `text data`, but the default varies according to your debug server.

Standard Example:

```
> load all
Download Options: text data bss
```

Advanced Example:

```
> load all noload bss
Download Options: text data
```

**m [-dsize] address[=val]**

If =*val* is not specified, the indicated memory *address* on the target is read.

If =*val* is specified, *val* to the specified memory *address* on the target is written.

Memory addresses and values must be specified in hexadecimal (with or without a leading 0x).

The optional **-d** argument can be used to set the access size to byte (**-d1**), short (**-d2**), or long (**-d4**). The default is **-d4**.

Examples:

```
> m 1000
7ca62b78
> m 1000=12345678
> m -d2 1000
1234
```

**nofail command**

Executes the specified *command* and always returns success.

**noload [all] [text] [data] [bss]**

If no argument is specified, the current **noload** settings are displayed.

If one or more arguments are specified, **noload** specifies which sections to exclude from host-to-target downloads. *.text* sections contain code, *.data* sections contain initialized variables, and *.bss* sections contain uninitialized data. Setting the **all** option excludes all of these sections from the download.

Use the command **noload bss** to exclude *.bss* sections from downloads. These sections are cleared to all zeros by programs compiled with Green Hills tools; therefore, downloading them to the target is usually unnecessary.

**open file**

Opens the specified *file* for writing and returns a file descriptor.

**print string**

Prints *string* with script variable expansion.

Example:

```
> print Test
Test
```

**random max**

Generates and returns a pseudo-random number between 0 and *max*-1.

**regnum** *reg*[=*val*]

If *=val* is not specified, the specified register, *reg* is read and returned.

If *=val* is specified, **regnum** writes *val* to the specified register, *reg*.

Registers are specified by MULTIregister number and values in hexadecimal with no leading 0x.

Example:

```
> regnum 0=deadbeef
> regnum 0
Register 0=deadbeef
```

**rg** *regname*[=*val*]

If *val* is not specified: Reads and returns the specified register, *regname*.

If *val* is specified: Writes *val* to the specified register, *regname*.

**run** [*address*]

If *address* is not specified, the processor is run from the current program counter(PC).

If *address* is specified, it sets the program counter to *address* and then runs the processor.

Use this command very carefully. It is possible to run a program on the board when MULTI thinks it is halted, which causes unpredictable results.

A common use for this command is to run a ROM monitor program so that the monitor can set up the board properly before MULTI downloads a program.

**script** *file*

Executes the commands in the specified script file, *file*, as if they were typed in line by line.

**setrange** *addr length*

Sets a checked address range beginning at *addr* and extending for *length* bytes.

A checked address range is a range of addresses that are considered inaccessible. Any memory access, read, or write to a checked address range can fail.

The **setrange** command is useful for restricting access to target memory that might be sensitive or volatile.

**setup** *file*

Specifies a script file, *file*, to be automatically run immediately prior to every host-to-target download.

**sleep** *n*

Suspends the debug server for *n* seconds.

### **status**

Prints and returns the current status of the debug server using the following status codes:

```
0 Running
1 Stopped by breakpoint
2 Single step completed
3 Exception
4 Halted
5 Process exited
6 Process terminated
7 No process
8 (Unassigned)
9 Stopped by hardware breakpoint
10 Failure
11 Process ready to run
12 Host system call in progress
13 Target reset
```

### **step**

Single steps the target CPU from the current program counter location.

### **undef *variable***

Removes *variable* and releases any memory associated with it.

Example:

```
> x=5
> undef x
> print $x
Error: variable undefined!
```

## **Additional Debug Server Commands**

The following table indicates where you can find information about additional commands (if any) available for your particular debugging interface.

| <b>Debugging interface</b>                                            | <b>Where to find additional commands for this target</b> |
|-----------------------------------------------------------------------|----------------------------------------------------------|
| Green Hills Debug Probe<br>(Green Hills Probe or<br>SuperTrace Probe) | <i>Green Hills Debug Probes User's Guide</i>             |



---

## The Green Hills Debug Server Scripting Language

---

In addition to the commands listed in “Generic Debug Server Commands” on page 46 and the additional commands available to your specific debug server (see “Additional Debug Server Commands” on page 52), a full scripting language is available from the debug server command line for all of the debug servers in this book except **rtserv** and the Simulator for PowerPC (**simppc**), and **cafeserv**.

You can run scripts by:

- Entering scripts one line at a time at the command prompt. When entering a script line by line, nothing is executed until the top-level enclosing **while** or **if** statement is terminated.
- Storing commands in a script file and then running the file using the **script** command.
- Storing commands in a script file and then using the **-setup** option or **setup** command (or the **Target Setup Script** field in some Connection Editors) to ensure that the script file is automatically executed immediately prior to every host-to-target download.

### General Notes

When writing scripts, keep the following points in mind:

- There can be no more than one statement per line.
- Any line that has the # character as the first non-whitespace character is treated as a comment.
- Variables do not need to be declared before they are used.
- Variable types are determined automatically. For example, an identifier that is bound to an integer variable can later be assigned to a string variable.
- Variable and function names are not case-sensitive.

## Scripting Syntax

The following sections describe and give examples of some features of the Green Hills debug server scripting language.

### Expressions

Expressions in the Green Hills debug server scripting language are similar to C language expressions. The following table contains, in order of precedence, the valid operators you can use in expressions. Note that a string is treated like an integer if it contains a string representation of an integer.

| Operator | Integer Function                                   | String Function                                    |
|----------|----------------------------------------------------|----------------------------------------------------|
| ( )      | Grouping of operators to ensure desired evaluation | Grouping of operators to ensure desired evaluation |
| *        | Multiplication                                     | Invalid                                            |
| /        | Division                                           | Invalid                                            |
| %        | Modulus                                            | Invalid                                            |
| +        | Addition                                           | Concatenation                                      |
| -        | Subtraction                                        | Invalid                                            |
| <<       | Bitwise left shift                                 | Invalid                                            |
| >>       | Bitwise right shift                                | Invalid                                            |
| <        | Less than                                          | Alphabetic less than                               |
| <=       | Less than or equal to                              | Alphabetic less than or equal to                   |
| >        | Greater than                                       | Alphabetic greater than                            |
| >=       | Greater than or equal to                           | Alphabetic greater than or equal to                |
| ==       | Equality test                                      | Equality test                                      |
| !=       | Inequality test                                    | Inequality test                                    |
| &        | Bitwise AND                                        | Invalid                                            |
| ^        | Bitwise XOR                                        | Invalid                                            |
|          | Bitwise OR                                         | Invalid                                            |
| &&       | Logical AND                                        | Invalid                                            |
|          | Logical OR                                         | Invalid                                            |

## Assignments

The syntax for an assignment is:

*identifier* = *expression*

The *expression* is evaluated and the result is stored as a variable with the given *identifier*. String, integer, and array variables are supported. Identifiers can contain alphanumeric characters and the underscore (\_) character, but cannot begin with a number.

## Arrays

Arrays are indexed lists of variables. Each cell in an array can contain a string, an integer, or an array. Array indexing begins with the index 0. An array can be created by assigning an entire array to an identifier or by assigning a string, an integer, or an array to one cell of an array. To reference a cell, follow the array identified by the index contained in square brackets. If an array cell contains another array, the elements in the second array are accessed by appending an additional index in square brackets. The following example demonstrates the various methods of array access.

```
endl="\n"
bar[3] = 42
foo = { bar, 7, "hello" }
print $foo[2] world.$endl
if(foo[0][3]==bar[2+1])
    print Array indexing works.$endl
endif
```

Arrays are dynamically allocated in a sparse fashion. For example, making an assignment to `foo[0]` and then to `foo[100]` only allocates two array cells, and no space is used for the undefined array cells 1 through 99. After an array element has been allocated, it can only be deallocated by using the **undef** command on the top-level array identifier.

## Conditionals

The following table explains the syntax for conditionals.

| Syntax                                                                                                                                                                                 | Effect                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>if</b> <i>expression</i><br><i>statements</i><br><b>endif</b>                                                                                                                       | If <i>expression</i> evaluates to zero, nothing happens.<br>Otherwise, the block of statements between the <b>if</b> and <b>endif</b> lines are executed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>if</b> <i>expression</i><br><i>statements</i><br><b>else</b><br><i>statements</i><br><b>endif</b>                                                                                   | If <i>expression</i> evaluates to zero, the debug server executes the block of statements between the <b>else</b> and <b>endif</b> lines.<br>Otherwise, the block of statements between the <b>if</b> and <b>else</b> lines are executed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>if</b> <i>expression1</i><br><i>statements</i><br><b>elif</b> <i>expression2</i><br><i>statements</i><br>[ <b>elif</b> <i>expressionX</i><br><i>statements</i> ]...<br><b>endif</b> | If <i>expression1</i> does not evaluate to zero, the debug server executes the block of statements between the <b>if</b> and <b>elif</b> lines.<br><br>If <i>expression1</i> does evaluate to zero and <i>expression2</i> does not evaluate to zero, the debug server executes the block of statements between the <b>elif</b> and <b>endif</b> lines.<br><br>If both <i>expression1</i> and <i>expression2</i> evaluate to zero, the debug server continues evaluating each of the subsequent <b>elif</b> expressions (if any) that occur before the <b>endif</b> statement and will execute the block of statement associated with each <b>elif</b> that does not evaluate to zero.<br><br>If none of the <b>elif</b> expressions evaluates to non-zero, the debug server will not execute anything. |

## Loops

The following table explains the syntax for loops.

| Syntax                                                                 | Effect                                                                                                                                                                                                                                                         |
|------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>while</b> <i>expression</i><br><i>statements</i><br><b>endwhile</b> | The statements between the <b>while</b> and <b>endwhile</b> lines are executed as long as <i>expression</i> does not evaluate to zero.<br><br>Be careful to avoid infinite loops. If an infinite loop occurs, you must shut down and restart the debug server. |

## Variable Expansion

To use script variables as arguments to Green Hills debug server commands, you must prepend the variable name with one or two \$ characters.

- To pass a variable to a command in its default text representation, prepend the variable name with a single \$ character. This passes a decimal string for integer variables and passes the string itself for string variables. An entire array cannot be given as an argument to a command.
- To pass an integer variable to a command as a hexadecimal string, prepend the variable name with two \$ characters. Use this method with commands such as **m** that require arguments in hexadecimal form.

Variable expansion must be unambiguous. The script parser attempts to use the longest legal identifier name following the \$ character. In the following example, the user has attempted to print the string `bar` after the expansion of the variable `foo`. The parser interprets this as printing the value of the variable `foobar` and reports that the variable is not defined:

```
>foo="foo"
>print $foobar
Error: variable undefined!
```

## Example Scripts

You can use the following examples of the Green Hills debug server scripting language as a guide when writing your own scripts.

### Example 1. Testing Script File Access in ASCII Mode

For this example, assume that the file **test.ascii** contains the following text:

```
This is a test of script file access in ascii mode.
This should print 3 lines of which this is the second.
And this is the third.
```

The following script commands access the file **test.ascii**:

```
file = open test.ascii
filecontents=""
totalchars=0
while(numlinechars=fread file line)
    filecontents=filecontents+line
    totalchars=totalchars+numlinechars
endwhile
close file
endl="\n"
print Read: $filecontents$endl
print Total of $totalchars characters read.$endl
```

The output of this example script on a UNIX host is:

```
Read: This is a test of script file access in ascii mode.  
This should print 3 lines of which this is the second.  
And this is the third.
```

```
Total of 130 characters read.
```

## **Example 2. Accessing a File**

The following script is another example of accessing a file:

```
i=100  
file = open temp.bin  
while(i>0)  
    fprintf file $i  
    i=i-1  
endwhile  
close file  
file = open temp.bin  
sum=0  
while(freadb file i)  
    sum=sum+i  
endwhile  
close file  
endl="\n"  
print The numbers between 1 and 100 sum to $sum!$endl
```

The output of this example script is:

```
The numbers between 1 and 100 sum to 5050!
```

## **Example 3. Calculating a CRC32 Value**

The following script calculates the CRC32 value of the memory range 0x010000 to 0x010100 and prints the result in the **Target** pane. This script demonstrates the use of loops, conditional statements, expressions, variables, and other debug server scripting constructs in a real application.

```
# Change the following values to specify the memory
# range you want to calculate a CRC32 for.
# Note: locations from memstart to memend-1 are used
# to compute the CRC32 value.
memstart=0x010000
memend=0x010100
# This is the CRC32 polynomial. This is the same as
# is used in ethernet packets.
p=2+4+16+32+128+256+1024+2048+4096+65536+4194304+8388608+67108864
r=0
ptr=memstart
while(ptr<memend)
    currbyte=m -d1 $$ptr
    currbit=128
    while(currbit)
        test=r&(1<<31)
        r=r<<1
        r=r|(currbyte&currbit)
        if(test)
            r=r^p
        endif
        currbit=currbit>>1
    endwhile
    ptr=ptr+1
endwhile
# This loop is for the 32 zeros appended to the
# original memory contents
i=0
while(i<32)
    test=r&(1<<31)
    r=r<<1
    if(test)
        r=r^p
    endif
    i=i+1
endwhile
# Now the resulting 32 bit CRC is in r
# Many of the same ASCII control codes that are used
# in C are supported in debug server scripts.
endl="\n"
print CRC32 = $$r$endl
```





# Index

---

## A

### **addressof** command

for legacy debug scripts, 47

### **amask** command

for legacy debug scripts, 47

### arrays

for legacy debug scripts, 55

### assignments

in legacy debug scripts, 55

## B

### **.bss** sections

legacy debug scripts, 49 to 50

## C

### cafeserv

connecting MULTI to, *see* Connection

#### Methods

### checked address range

for legacy debug scripts, 47, 51

### **close** command

for legacy debug scripts, 47

### command help, 6

### commands

help regarding, 6

**helpview**, 6

legacy debug scripts, 18, 46

**addressof**, 47

**amask**, 47

**close**, 47

**debug**, 47

**delrange**, 47

**delrangeall**, 47

**fprint**, 47

**fprintb**, 47

**fread**, 48

**freadb**, 48

**getenv**, 48

**halt**, 48

**help**, 48

**listrange**, 48

**listvars**, 49

**load**, 49

**m**, 50

**nofail**, 50

**noload**, 50

**open**, 50

**print**, 50

**random**, 50

**regnum**, 51

**rg**, 51

**run**, 51

**script**, 51

**setrange**, 51

**setup**, 51, 53

**sleep**, 51

**status**, 52

**step**, 52

**undef**, 52

MULTI Debugger, 18

**memread**, 24

**memwrite**, 24

**reset**, 24

**wait**, 24

conditionals

# Index

---

- for legacy debug scripts, 55
- configuration options
  - help regarding, 6
- connecting MULTI to your target, iv
  - See also* **Connection Editor**;
  - Connection Methods
  - list of supported PowerPC targets, v
  - native targets, v
  - overview, iv to v

## **Connection Editor**

- Cafe OS (**cafeserv**), 39
- Connection Methods
  - creating
    - Cafe OS (**cafeserv**), 38
  - editing
    - Cafe OS (**cafeserv**), 39
- context-sensitive help, 6
- conventions
  - typographical, viii

## **D**

- .data** sections
  - legacy debug scripts, 49 to 50
- debug** command
  - for legacy debug scripts, 47
- debug servers
  - commands, *see* commands
  - debugging interfaces supported by, 21
  - installing, 14
  - options, *see* options
  - role of, iv
- debugging interfaces
  - debug servers associated with, 21
  - list of supported, v
- delrange** command
  - for legacy debug scripts, 47
- delrangeall** command
  - for legacy debug scripts, 47
- document set, v, vii

## **E**

- early MULTI board setup scripts
  - using debugger hooks, 34
- echo** command
  - for legacy debug scripts, 47
- examples
  - legacy debug scripts, 57
- expressions
  - legacy debug scripts, 54

## **F**

- F1 key, for help, 6
- fprint** command
  - for legacy debug scripts, 47
- fprintb** command
  - for legacy debug scripts, 47
- fread** command
  - for legacy debug scripts, 48
- freadb** command
  - for legacy debug scripts, 48

## **G**

- getenv** command
  - for legacy debug scripts, 48

## **H**

- halt** command
  - for legacy debug scripts, 48
- hardware, target, *see* targets
- help, *see* online help
- help** command
  - for legacy debug scripts, 48
- Help** menu
  - online help, obtaining, 6
- Help Viewer
  - navigating, 8 to 10
  - opening additional manuals in, 11
  - window, 7
- helpview** command, 6

# Index

---

## I

installation  
    debug server, 14  
    MULTI, 14  
    target hardware, 14  
Integrated Development Environment (IDE), *see* MULTI Integrated Development Environment (IDE)  
INTEGRITY Real-Time Operating System (RTOS), v  
interrupts  
    disabling sources during target setup, 23

## L

Launcher, *see* MULTI Launcher  
legacy debug scripts  
    arrays, 55  
    assignments, 55  
    .bss sections, 49 to 50  
    conditionals, 55  
    .data sections, 49 to 50  
    examples, 57  
    expressions, 54  
    loops, 56  
    registers, reading and writing, 51  
    running, 53  
    .text sections, 49 to 50  
    uses for, 53 to 54  
    variables  
        declaring, 53  
        expansion, 56  
        type, 53  
    writing, 53  
legacy debug server setup (.dbs) scripts, *see* setup scripts, board  
linker directives files  
    created by **Project Wizard**, 14 to 15, 17  
    creating custom, 17

    editing, 30

**listrange** command  
    for legacy debug scripts, 48  
**listvars** command  
    for legacy debug scripts, 49  
**load** command  
    for legacy debug scripts, 49  
loops  
    for legacy debug scripts, 56

## M

**m** command  
    for legacy debug scripts, 50, 57  
memory, target  
    configuring using a setup script, 22  
    testing access through your debugging interface, 20  
**memory.ld** file  
    editing, 30  
**memread** command  
    for MULTI Debugger, 24  
**memwrite** command  
    for MULTI Debugger, 24  
MULTI  
    installing, 14  
MULTI board setup (.mbs) scripts, *see* setup scripts, board  
MULTI Integrated Development Environment (IDE)  
    document set, vii  
    online help, 6  
    overview, 2  
MULTI Launcher  
    overview, 4

## N

native targets  
    connecting MULTI to, v  
**nofail** command  
    for legacy debug scripts, 50

# Index

---

**noload** command  
for legacy debug scripts, 50

## O

online help, 6  
    *See also* Help Viewer  
for commands, 6  
for configuration options, 6  
context-sensitive, 6  
limitations, 10  
online manuals, 6  
overview, 6

**open** command  
for legacy debug scripts, 50  
OS targets, v

## P

**print** command  
for legacy debug scripts, 50  
Project Manager  
    configuring target hardware with, 14  
    testing target configuration with, 17  
**Project Wizard**  
    configuring target hardware with, 15,  
    17  
    testing target configuration with, 14

## R

**random** command  
for legacy debug scripts, 50  
registers, reading and writing  
    legacy debug scripts, 51  
    testing, 19  
**regnum** command  
for legacy debug scripts, 51  
**reset** command  
for MULTI Debugger, 24  
**rg** command  
for legacy debug scripts, 51  
**run** command

for legacy debug scripts, 51  
running legacy debug scripts, 53

## S

**script** command  
for legacy debug scripts, 51, 53  
scripting language  
    MULTI board setup (**.mbs**) vs. legacy  
    debug server setup (**.dbs**) style, 18  
    new (**.mbs**) vs. old (**.dbs**) style, 31  
**setrange** command  
for legacy debug scripts, 51  
**setup** command  
for legacy debug scripts, 51, 53  
**-setup** option  
for all debug servers, 33, 53  
setup scripts, board  
    created by **Project Wizard**, 14 to 15,  
    17, 21  
    creating custom, 17, 21  
    editing, 21 to 22  
    example, 25 to 26  
    MULTI board setup (**.mbs**) vs. legacy  
    debug server setup (**.dbs**) style, 18  
    new (**.mbs**) vs. old (**.dbs**) style, 31  
    running **.dbs** scripts manually, 34  
    running **.dbs** scripts when  
        connecting, 33  
    running **.mbs** scripts manually, 33  
    running **.mbs** scripts when  
        connecting, 32  
    specifying and running, 31  
    specifying in **Connection Editor**, 19  
    testing commands for, 21  
**setup=** option for **connect**  
    general usage, 32  
simulators  
    supported by MULTI, v  
**sleep** command  
for legacy debug scripts, 51

# Index

---

**standalone\_config.ld** file

editing, 30

**status** command

for legacy debug scripts, 52

**step** command

for legacy debug scripts, 52

system requirements, *see* targets

## T

target connections, v

*See also* connecting MULTI to your target

targets

configuring, 14 to 15, 17, 31

*See also* setup scripts, board

configuring memory using a setup script, 22

connecting MULTI to, *see* Connection Methods

debug servers for supported, 21

installing, 14

list of supported, v

native, v

OS, v

testing the configuration of, 18 to 20

.text sections

legacy debug scripting, 49

legacy debug scripts, 50

ThreadX targets, v

Tornado targets, v

typographical conventions, viii

## U

**undef** command, 55

for legacy debug scripts, 52

## W

**wait** command

for MULTI Debugger, 24

workspaces

overview, 5

writing legacy debug scripts, 53